(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 524 773 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **92306537.9**

(51) Int. Cl.[5] : **G06F 9/44**

(22) Date of filing : **16.07.92**

(30) Priority : **23.07.91 US 734397**

(43) Date of publication of application :
27.01.93 Bulletin 93/04

(84) Designated Contracting States :
DE FR GB

(71) Applicant : **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor : **Cook, John Andrew**
7701 Jester Boulevard
Austin, Texas 78750 (US)
Inventor : **Vepstas, Linas Lukas**
1518 Enfield Road
Austin, Texas 78703 (US)

(74) Representative : **Blakemore, Frederick Norman**
IBM United Kingdom Limited Intellectual Property Department Hursley Park
Winchester Hampshire SO21 2JN (GB)

(54) Multiple command set support for rendering components.

(57) A method and apparatus for supporting multiple command sets in a single rendering adapter. A device driver operating in tandem with a rendering adapter and its associated microcode interprets disparate command sets without separate control/interpretation sections being maintained in the rendering adapter. Rendering adapter microcode in the adapter for supporting a first command set is extending to support commands in a second command set not capable of being mapped in the first command set. The device driver receives the disparate data stream command sets generated by application programs and destined for the rendering adapter. If a first command set command is received by the device driver, it is grassed on to the rendering adapter substantially unmodified. If a second command set command is received by the device driver, an attempt is made to map the command into a first command set command, and then transfer this mapped command to the rendering adapter. If no mapping is possible, the received command is reformatted to conform to the syntax of the first command set, and then sent to the rendering adapter. As the rendering adapter microcode has been extending beyond interpreting and executing the first command set, this reformatted second command set command can be processed by the same parser resident in the rendering adapter microcode.

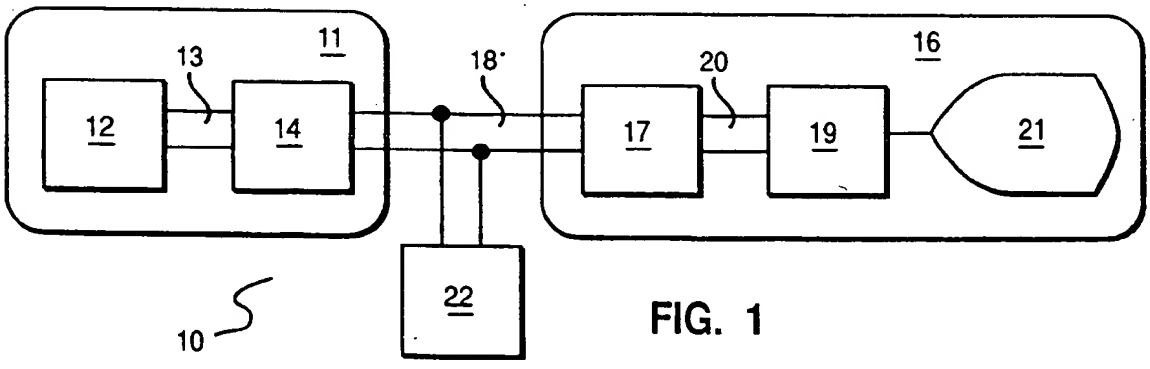

FIG. 1

EP 0 524 773 A1

This invention pertains to the field of data processing systems, and more specifically to the construction and interpretation of command data streams used to control rendering components in a data processing system.

Various data processing rendering components are known in the prior art. Such components translate a series of instructions/commands into specific rendering operations. A graphics adapter acts as an interface between a computer and a computer display device. The graphics adapter accepts drawing commands from a data processing computer or microprocessor, translates these commands, and generates the appropriate driving signals used to drive and display an image on a computer display. A printer adapter similarly accepts printing commands, interprets these commands, and generates the driving signals used to actuate the printer's electrical, electromagnetic, and/or laser-controlled print engine.

These types of adapters are designed to accept a known and defined syntax or command set. Examples of such graphics command sets are PHIGS, GL, and GKS. These command sets contain the actual drawing commands used to generate graphical displays on a data processing systems' display device. Examples of data streams sent to a rendering printer adapter are Postscript® (Trademark of Adobe Systems, Inc.) and IPDS. The rendering commands sent to a rendering adapter are generated from some type of command generator. This command generator could be, for example, a drawing/rendering application program running on a data processing computer. Such an application program could be running either locally within the same processing system which contains the graphics adapter, or remotely where the application program is running on a processing system separate from the system containing the graphics adapter and display. In the remote configuration, drawing commands would be transmitted from the application processing system to the drawing system via conventional telecommunications techniques. Various command sets have evolved over the years which correspond to changes in graphics adapter technology. For example, early command sets supported 2-dimensional graphics adapters, which were only capable of displaying 2-dimensional objects. Technological improvements in electronic packaging density have resulted in 3-dimensional graphics adapters being introduced into the marketplace. As existing 2-d command sets could not take advantage of the new functionality contained

Within these 3-d graphics adapters, new command sets were created to support these 3-d graphics adapters. In addition, differing 3-d command sets have evolved which conform to various industry defacto and international standards. These graphics adapters have generally been designed to handle a single type of command set. This single-command set support was required to maximize the performance of the graphics adapter's ability to accept drawing commands and render a drawing on the processing system display or printer. This single-command set support was also desired in order to minimize system costs, as the number and complexity of rendering adapter components increases substantially if more than a single command set were to be supported. This is due to increases in control circuitry and/or rendering adapter microcode required to detect and interpret multiple command set data streams. In general, a given user environment does not want the increased overhead and associated performance degradation when the processing system actually being used is generating a single command set. Providing a multi-command set function causes an end-user to unnecessarily shoulder an increase in system overhead and cost.

However, some end-users do have a need for supporting multiple command sets, and are willing to pay the costs and absorb the performance degradations associated with such multi-command set support To meet this demand of supplying multiple command set functionality, prior art systems have provided two distinct command set interpreters in the rendering adapter, one for each respective command set. This has resulted in a substantial increase in cost over and above a single command set rendering adapter. This increase in cost results from increases in volatile and/or nonvolatile data processing memory and other control electronics used to switch between command sets efficiently in order to minimize the resultant system performance degradation.

In accordance with the present invention, there is now provided a method for providing multi-command support in a rendering adapter, comprising:

receiving first commands of a first command set by an intermediate driver and providing the first commands to a rendering adapter; and

receiving second commands of a second command set by the intermediate driver, converting any of the second commands into resulting first command set commands and providing the resulting first command set commands to the rendering adapter.

The present invention provides a method and apparatus for supporting multiple command sets in a single rendering adapter. The present invention therefore provides a system which can support multiple command sets for a rendering adapter in a cost/performance efficeint manner. In accordance with the present invention, there is provided a framework which can interpret either and/or both command sets with a single control/interpretation section contained within the rendering adapter. Therefore, there is no requirement for separate adapter control/interpretation sections on the adapter for each respective command. set. This advantageously results in a decrease in adapter complexity, associated adapter

memory requirements, and in overall adapter costs. Multiple-command set system performance comparable to that of single-command set systems is also maintained.

Viewing a second aspect of the present invention, there is now provided a method for providing multi-command support in a rendering adapter, comprising: generating any of first command set commands and second command set commands by an application program executing on a data processing system; initiating a transfer, by the application program, of any of first command set commands and second command set commands to a rendering adapter; receiving first commands of a first command set by an intermediate driver; providing the first commands to a rendering adapter; receiving second commands of a second command set by the intermediate driver, wherein the intermediate driver converts the second commands into resulting first command set commands; and providing the resulting first command set commands to the rendering adapter.

Viewing a third aspect of the present invention, there is now provided a system for providing multi-command support in a rendering component, comprising: means for receiving first commands of a first command set and second commands of a second command set by an intermediate driver, wherein the intermediate driver converts any of the second commands into resulting first command set commands; means for providing any of the first commands, the second commands, and the resulting first command set commands to the rendering adapter; and means for processing any of the first commands, the second commands, and the resulting first command set commands by the rendering adapter.

Viewing a fourth aspect of the present invention, there is now provided a system for providing multi-command support in a rendering adapter, comprising: means for receiving first commands of a first command set and second commands of a second command set by an intermediate driver; means for providing the first commands, by the intermediate driver, to a rendering adapter; means for converting, by the intermediate driver, the second commands which are unique to the second command set into resulting first command set commands; and means for providing the resulting first command set commands and any non-unique second commands, by the intermediate driver, to the rendering adapter.

Viewing a fifth aspect of the present invention, there is now provided a method for controlling a rendering component of a data processing system, comprising: receiving rendering commands by the rendering component; interpreting and executing the rendering commands by a single process; and maintaining a first state area for first state commands and a second state area for second state commands.

Viewing a sixth aspect of the present invention, there is now provided an apparatus for controlling a rendering component of a data processing system, comprising: means for receiving rendering commands by the rendering component; means for interpreting and executing the rendering commands by a single process; and means for maintaining a first state area for first state commands and a second state area for second state commands.

In a preferred embodiment of the present invetnion, a device driver operates in tandem with a rendering adapter and its associated microcode. A first command set is defined which represents the largest command set being supported in the system. A second command set is defined which represents a second command set being supported. Rendering adapter microcode in the adapter for supporting the first command set is extended to support commands in the second command set not common to, or capable of being mapped into, the first command set.

The device driver receives disparate data stream command sets generated by application programs and destined for the rendering adapter. If a first command set is received by the device driver, it is passed on, or transmitted, to the rendering adapter substantially unmodified. If a second command set command is received by the device driver, an attempt is made to map the command into a first command set command, and then pass this mapped command to the rendering adapter as a resulting first command set command. If no mapping is possible, i.e. no equivalent command exists in the first command set or the command is not common to both command sets, then the received command is reformatted to conform to the syntax of the first command set, and contains the received second command set unique command internal to this reformatted package. This reformatted second command set unique command is then sent to the rendering adapter as a resulting first command set command. As the rendering adapter microcode has been extended beyond interpreting and executing the first command set, this reformatted second command set unique command can be processed by the same parser resident in the rendering adapter microcode. In this fashion, the microcode residing on the rendering adapter only needs to parse and route a single data stream syntax.

As to the rendering adapter microcode, a first state is defined for the first command set to be supported in the multi-command set adapter. The state refers to such things as color, normal, positions, stacks of pick-identifying names, pick data, and stacks of matrices. A second state is defined for the second command set to be supported. A special command is added to the adapter command set to allow switching between these first and second states. For rendering commands which are common to either command set, this command is used to direct the commands to use the proper state variables.

Resource control commands contained within the two command sets were similarly modified to present a single, uniform data stream to the rendering adapter. As a result, only a small subset of new resource control command functionality was added to the adapter microcode to support the additional functionality contained in the second command set.

The resulting system reduces the overall system complexity, the amount of adapter microcode required, and system cost while maintaining performance equivalent to a single command set system. It will be appreciated that the present invention can be used for a wide variety of rendering adapters including, but not limited to, graphics, printer, multimedia, and audio.

Preferred embodiments of the present invention will now be described with reference to the accompanying drawings in which:

Figure 1 shows the interplay between a command generator, rendering component, and rendering device.

Figure 2 shows a remote command generator.

Figure 3 shows prior art command sets.

Figure 4 shows a unified command set built from disparate command sets.

Figures 5a-5b detail the internal operation of the intermediate driver.

Figure 6 details the building of a shell command by the intermediate driver.

Figure 7 details the operation of expanding a single command into multiple commands.

Figure 8 details the internal operation of the rendering component.

Figure 9 is a overall system block diagram.

Figure 1 shows a data processing system 10 and the interplay between an application process 11 comprising a command generator 12 and an intermediate driver 14, and a rendering component 16. The command generator 12 could be an application program executing on a computer. An intermediate driver 14 receives rendering commands destined for the rendering device 16. In the preferred embodiment, intermediate driver 14 is a subroutine library which services the appropriate rendering component 16. The intermediate driver 14 manipulates and combines disparate incoming command sets and produces a single command set data stream. This resultant single command set data stream is then transferred to the rendering component 16. In the preferred embodiment, this rendering component is a graphics adapter. However, as will be understood by those skilled in the art, the concepts herein disclosed would similarly apply to other types of rendering devices, such as printer adapters, multimedia adapters, and audio adapters.

The interface 13 between the command generator 12 and the intermediate driver 14 is a subroutine calling interface, or API, in the preferred embodiment. An alternate embodiment would have a client/server interface, having a communication interface 13 between a client 12 and a server 14. In this alternate embodiment, the application program 12 transmits messages or commands to a receiving process 14 via some mechanism, such as pipes, sockets, streams, shared memory, or semaphores, as is generally known to those of ordinary skill in the art. The output of the intermediate device 14 is a series of commands which are transferred along a channel, bus, or communication link to a rendering adapter 16 via conventional transfer methods 18. Such methods include I/O port addressing, memory mapped I/O, and the like. These commands can contain various kinds of information, such as resource control commands, rendering instructions, or data.

The rendering device 16 comprises various subassemblies in the preferred embodiment. Adapter microcode executing on a general purpose digital signal processor (DSP) and hardware/firmware embodied in special purpose gate arrays (bus interface chips) provide the interface function at 17. The DSP connects to a rendering subsystem 19 using conventional internal wiring 20. The rendering subsystem 19, which provides rasterization in the preferred embodiment, comprises application specific VLSI chips, banks of VRAM/DRAM memory components, and digital-to-analog (DAC) chips. The rendering subsystem 19 drives a CRT display 21 in the preferred embodiment. However, there could similarly be a print head at 21 in an alternate embodiment, with the rendering subsystem 19 further comprising control for paper manipulation using conventional electo-mechanical components and techniques.

A rendering context manager 22 is similarly situated on the transfer means 18. The rendering context manager is a separate process which allows for multiple applications to use a single rendering adapter, by providing conventional multiprocessing support.

Figure 2 shows another embodiment of the invention wherein a command generator/driver subsystem 11 is located in a data processing system 24 distinct from the processing system 28 containing the rendering component 16. Conventional communications methods 38 are used to connect, or establish a session, between communication interfaces 30 and 40. These communication interfaces connect to the rendering device 34 at 36, and the command generator 11 at 42. It should be understood that the communication interfaces at 30 and 40 may be simple communication adapters, or may be more sophisticated subsystems having computer controlled queuing, scheduling, etc. as is commonly known in the art of remote data processing and internetworking.

Figure 3 indicates how prior art systems processed differing command sets. Each distinct command set 44 and 46 are processed by separate and distinct processes on a rendering adapter. The present invention described herein merges these disparate com-

mand sets 44 and 46 into a single command set having various components contained within. As shown in Figure 4, a single command set 49 results from merging the differing command sets 44 and 46 of Figure 3. First command set commands 46 and second command set commands 44 are merged, with commands common to both command sets being shown at 48. These common commands exist in both the first command set 46 and the second command set 44.

Figure 5a details the operation of the combined process 11 of Figure 1. This combined process consists of an application program or command generator 12, an intermediate driver 14 and their interconnection via a subroutine calling interface 13 (see Figure 1). When certain commands generated by 12 are known to be a part of the first command set 46 of Figure 4, such commands cause control to be passed to 60 of Figure 5a, where the command is transferred on to the rendering component 16 of Fig. 1 without modification. It will be understood that any slight modifications to the command would also be possible and not deviate from the scope and spirit of this invention. For example, a bit could be turned on in a defined field indicating the command had been queried and left unmodified by the device driver. The primary distinction here is that the general content of the received command does not need to be substantially modified prior to passing the command to the rendering component 16 of Figure 1.

Likewise, certain commands generated by 12 are known to be a part of the second command set 44 of Figure 4. Such commands cause control to be passed to 58 (the internal function to be later described in more detail), where the received command is embedded within a shell structure having syntax which conforms to a first command set. Control then passes to 60, where the resulting first-command-set-syntax-conforming command is transferred to the rendering component 16 of Fig. 1.

Likewise, certain commands generated by 12 are known to be mappable into an equivalent first command set command 48 of Fig. 4. If so, control passes to 56 where the command is mapped to a resulting first command set command. The mapping is accomplished using standard programming techniques by building the equivalent first command set command and placing parameters in the locations defined for the first command set command. The resulting first command set command is then transferred to the rendering component at 60.

Likewise, certain commands generated by 12 are known to be convertible to combinations of a first and a second command set. If so, control passes to 54 where a significant amount of processing is done to convert the incoming command into one or more of the commands 44, 46, and/or 48 (the internal operation to be later described herein). After such processing and conversion is complete, control is passed directly to 60, or to 56 or 58, where relatively minor mapping and encapsulation is performed before passing control to 60.

Resource control commands contained within the two command sets are similarly modified as described above to present a single, uniform data stream to the rendering adapter. As a result, only a small subset of new resource control command functionality is needed in the adapter microcode to support the additional functionality contained in the second command set.

The conceptual model for performing the above is shown in Fig. 5b. Commands are received from the command generator at 50. A determination is made at 51 as to whether this received command is a part of the first command set (46 of Fig. 4) or a common command (48 of Fig. 4). If so, processing continues at 60 where the command is transferred to the rendering component without modification. If the query at 51 indicates the received command is a part of the second command set (44 of Fig. 4), processing continues to 52, where a determination is made on whether the received command can be mapped into an equivalent first command set command 48 of Fig. 4. If so, control passes to 56 where the command is mapped to a resulting first command set command. This resulting first command set command is then transferred to the rendering component at 60. If the query at 52 indicates the received command cannot be mapped into an equivalent first command set command, it is determined at 53 if the received command can be converted to combinations of a first and a second command set commands. If so, this conversion is accomplished at 54, after which the resulting combination of commands are processed by 56, 58 or 60 as described above. If the determination at 53 indicates this is a unique second command set command 44 of Fig. 4, processing at 58 embeds the received command within a shell structure having syntax which conforms to a first command set command. Processing at 60 then passes this resulting first command set command (the unique second command set command with shell structure) on to the rendering component. It should be reiterated that the above is a conceptual model of the internal process, which can be implemented in numerous ways using programming techniques commonly known, such as If-Then-Else logic, Switch statements, Case statements, or table-lookup techniques.

The function performed by 58 of Figs. 5a and 5b will now be described in reference to Figure 6. As previously described, the command process 11 comprises a command generator 12 coupled to an intermediate driver 14 by transfer means 13. The command generator 12 generates a series of commands 120-128 destined for a rendering adapter 16. To exemplify the building of a shell command as a result of a series of individual commands, a display_polyline command

is shown. Second command set commands (44 of Fig. 4) are generated by the command generator 12. A begin_line command 120 is initially generated, followed by a series of vertex commands 122, 124 and 126 having vertex coordinates, and ending with an end_line command 128. When the begin_line command 120 is received by the intermediate driver 14, a shell structure as indicated at 130 is created. This shell structure will be ultimately passed on to the rendering adapter 16. The shell structure is initialized with a length at 132, an opcode at 134 indicating a polyline command, the number of points contained within the polyline at 136, and flags at 138. As each subsequent vertex command 122, 124, and 126 is received by the intermediate driver 14, the coordinate data contained with the vertex command is placed in the shell structure at 140, 142, and 144 respectively. The length 132, number of points 136, and flags 138 are also updated when inserting the vertex data. Finally, when the end_line command 128 is received by the intermediate driver 14, the processing in block 58 of Figure 4 terminates, and control passes to block 60 of Figure 4, where the shell structure 130 (resulting first-command-set-syntax-conforming command) is transferred via 18 to the rendering component 16. Other second command set commands 58 can be processed in a similar manner, using programming techniques generally known to those of ordinary skill in the art.

The function of block 54 of Figure 5a will now be described. This block converts received commands into a combination of first and second command set commands. Referring to Figure 7, a sample control flow is shown which exemplifies the function of block 54. In this example, a set_pattern command is received at 150. This results in block 54 searching in system memory at 152 to find a pattern as previously defined by a define_pattern command. A setup_load_attributes command is generated at 154, (which is a second command set command 44 of Fig. 4) to obtain the pattern attributes. A write_attributes command is then generated at 156 (which is also a second command set command 44 of Fig. 4) to write the pattern and its size to the rendering component 16. Finally, a set_interior_style command is generated at 158 (which is a first command set command 58 of Fig. 4), which instructs the rendering adapter to use this pattern as the interior style to be used in its subsequent rendering operations. Program control is returned at 160. The above description exemplifies the operation of block 54 in combination with blocks 56 and 58 of Fig. 5a, and how a single command can cause varying combinations of first and second command set commands to be issued to the rendering component. It will be understood to those of ordinary skill in the art that other commands received at 54 could be similarly processed using these and other conventional programming techniques. Upon block 54's completion, control can either pass directly to 60 (Fig. 5a) if no further processing is required, or to either block 56 or 58 if processing as previously described for these blocks is required (e.g. mapping a 48 command or encapsulating a series of second command set commands 44 into a first command set command 46).

Turning now to Figure 8, the rendering component operation will be described. In the preferred embodiment, the primary control for the rendering component is a microprocessor or microcontroller having microcode 74 associated therewith. This microcode is conceptually divided into three components. Commands unique to the first command set are processed by microcode 80. Commands unique to the second command set are processed by microcode 84. Commands which are common(i.e. non-unique commands) to both the first and second command sets are processed by microcode 82 in the preferred embodiment. Alternate embodiment could perform this via a VLSI gate array. When commands are received by the rendering component at 76, a query is made at 78 to determine the appropriate microcode 80, 82, or 84 to invoke. Query 78 is also used to control local memory 92 as follows. A first state 86 is defined for the first command set to be supported in the multi-command set adapter. The state refers to such things as color, normal, positions, stacks of pick-identifier names, pick data, and stacks of matrices, for example. A second state 88 is defined for the second command set to be supported. A special command is added to the adapter command set to allow switching at 90 between these first and second states. For rendering commands which are common to either command set 82, this command is used to direct the commands to use the proper state variables 86 or 88.

Figure 9 details the overall system environment 100. A keyboard 102, central processing unit 104, random access storage 106, a nonvolatile storage device 108, and a rendering adapter 110 attached to a rendering output device 112 are interconnected via a bus 114. The hardware devices listed may or may not further include appropriate adapters required for direct bus attachment. In an alternate embodiment, the keyboard 102 and/or random access storage 106 may attach directly to the central processing unit 104 in order to achieve increased performance. In the preferred embodiment, the rendering output device 112 is a display and the rendering adapter 110 is a graphics adapter. In an alternate embodiment, the rendering output device 112 is a printer, and the rendering adapter 110 is a printer adapter. Likewise, in an alternate embodiment, the rendering adapter 110 may attach directly to the CPU 104 in order to achieve enhanced performance.

Claims

1. A method of providing commands to a rendering adapter, comprising:
receiving first commands of a first command set in an intermediate driver;
transferring the first commands from the intermediate driver to a rendering adapter;
receiving second commands of a second command set in the intermediate driver;
converting, in the intermediate driver, the second commands which are unique to the second command set into resulting first command set commands; and
transferring the resulting first command set commands and any non-unique second commands from the intermediate driver to the rendering adapter.

2. A method as claimed in Claim 1 wherein the intermediate driver is a device driver.

3. A method as claimed in Claim 2 wherein the transferring is achieved by using memory-mapped I/O.

4. A method as claimed in Claim 3 wherein the transferring is achieved by using I/O port addressing.

5. Apparatus for providing commands to a rendering adapter, comprising:
An intermediate driver having means for receiving first commands of a first command set and second commands of a second command set and means for converting the second commands which are unique to the second command set into resulting first command set commands;
means for transferring the first commands, from the intermediate driver, to a rendering adapter; and
means for transferring the resulting first command set commands and any non-unique second commands, from the intermediate driver, to the rendering adapter.

6. Apparatus as claimed in Claim 5 wherein the rendering adapter is a graphics adapter.

7. Apparatus as claimed in Claim 5 wherein the rendering adapter is a printer adapter.

8. A data processing system comprising:
a keyboard;
a central processing unit;
memory means for storing data;
nonvolatile storage;
a rendering output device; and
a rendering adapter having means for receiving first commands of a first command set and second commands of a second command set by an intermediate driver, wherein the intermediate driver converts any of the second commands into resulting first command set commands, and means for transferring any of the first commands, the second commands and the resulting first command set commands to the rendering adapter.

9. A method for providing multi-command support in a rendering adapter, comprising:
receiving first commands of a first command set by an intermediate driver and providing the first commands to a rendering adapter; and
receiving second commands of a second command set by the intermediate driver, converting any of the second commands into resulting first command set commands, and providing the resulting first command set commands to the rendering adapter.

10. A system for providing multi-command support in a rendering adapter, comprising:
means for receiving first commands of a first command set and second commands of a second command set by an intermediate driver, wherein said intermediate driver converting any of said second commands into resulting first command set commands; and
means for providing any of said first commands, said second commands and said resulting first command set commands to said rendering adapter.

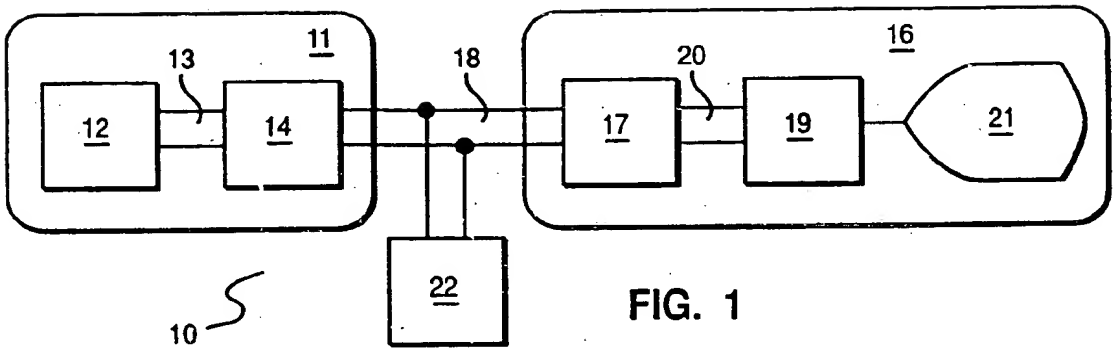

FIG. 1

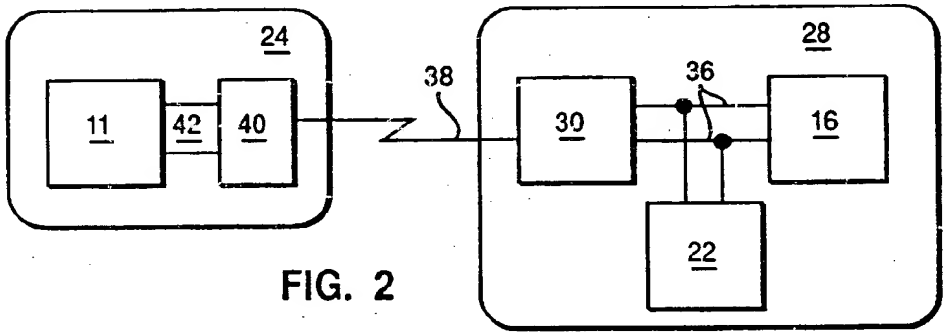

FIG. 2

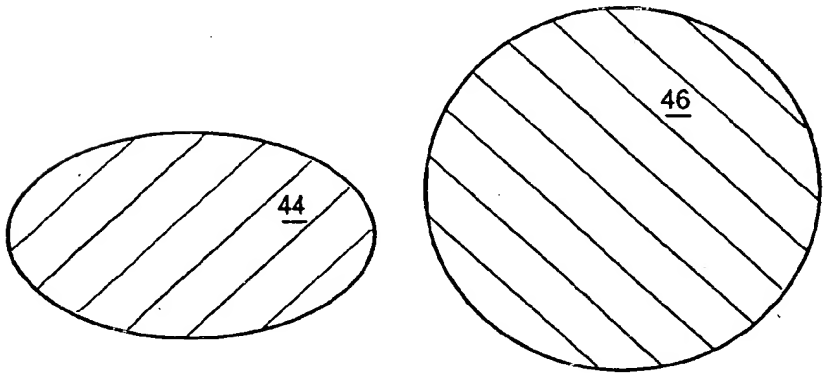

FIG. 3 PRIOR ART

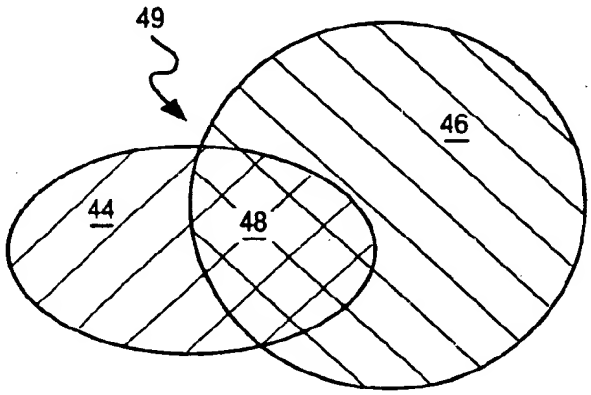

FIG. 4

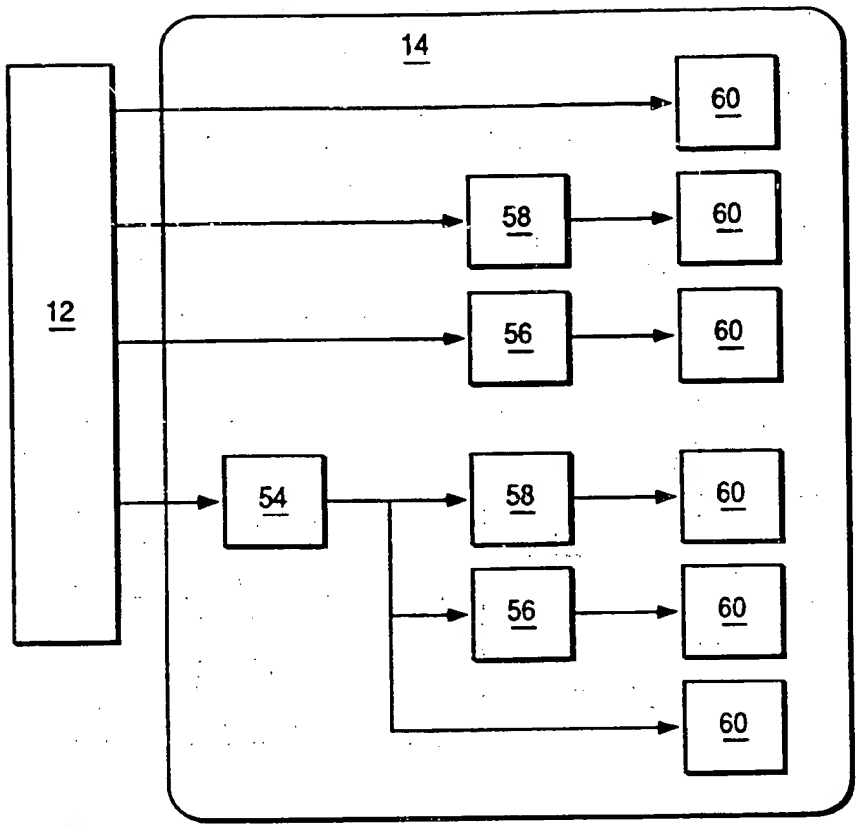

FIG. 5a

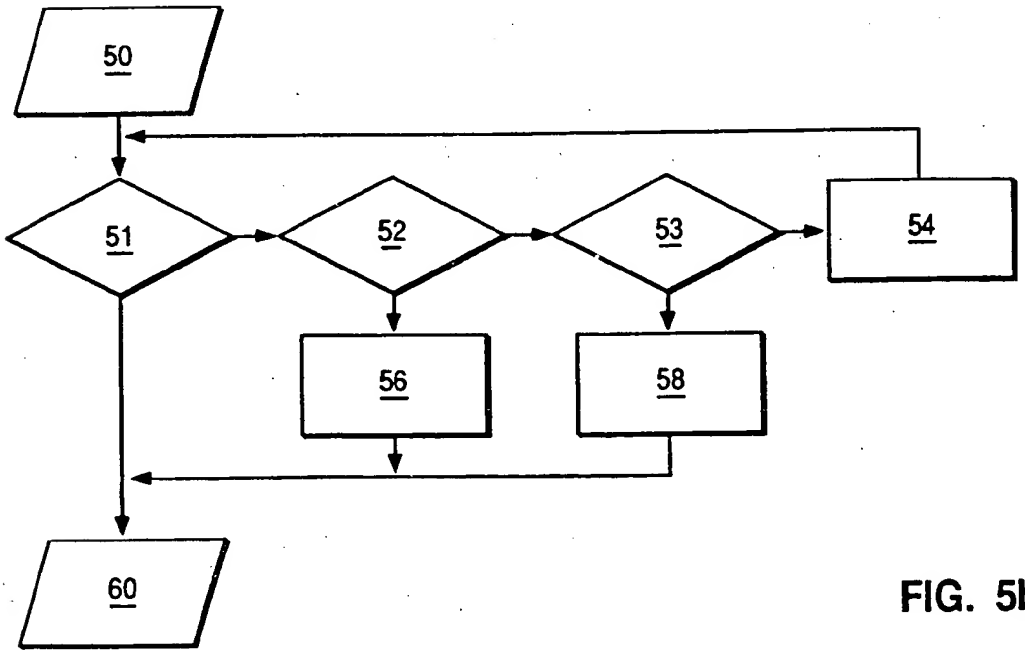

FIG. 5b

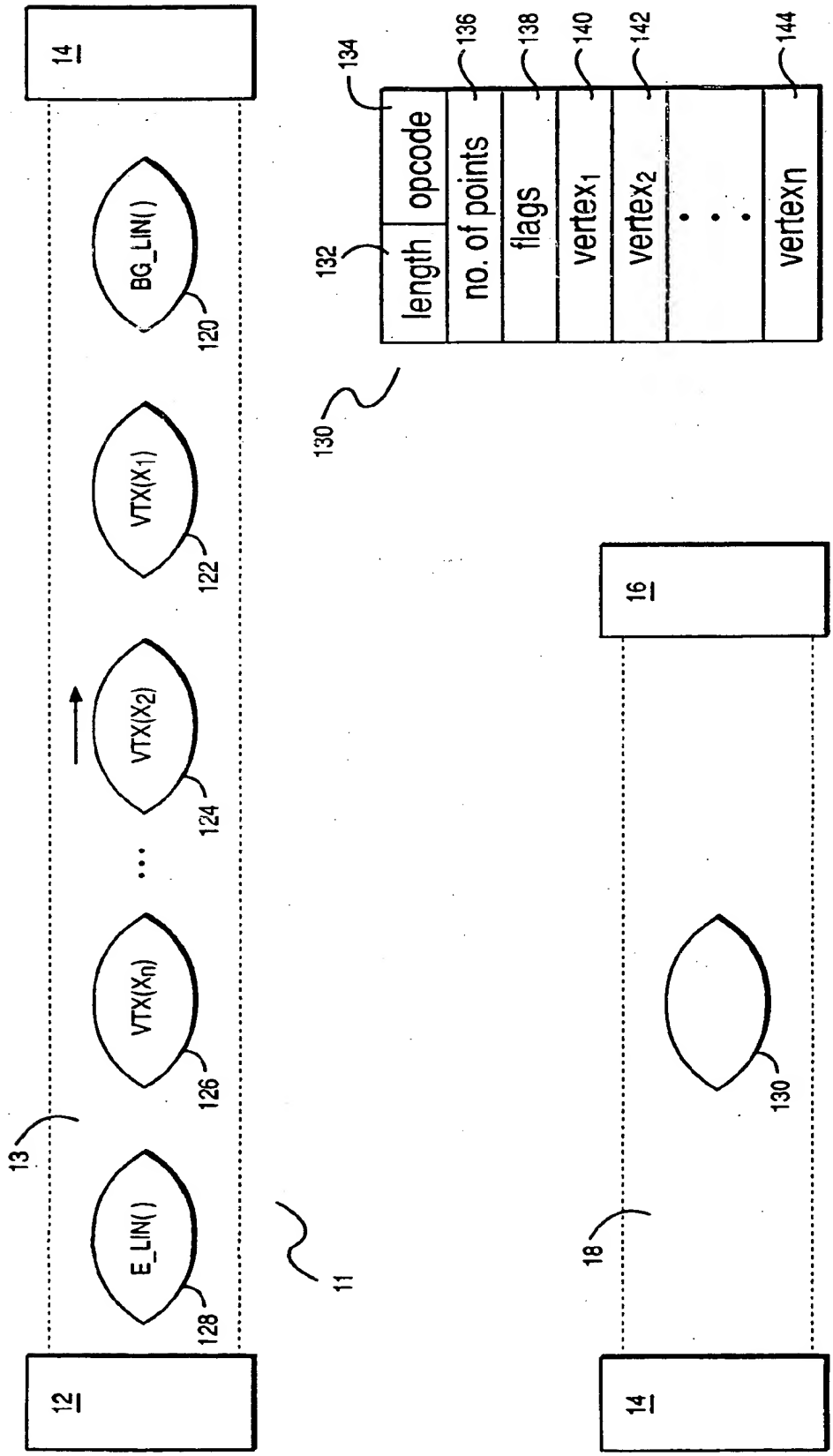

FIG. 6

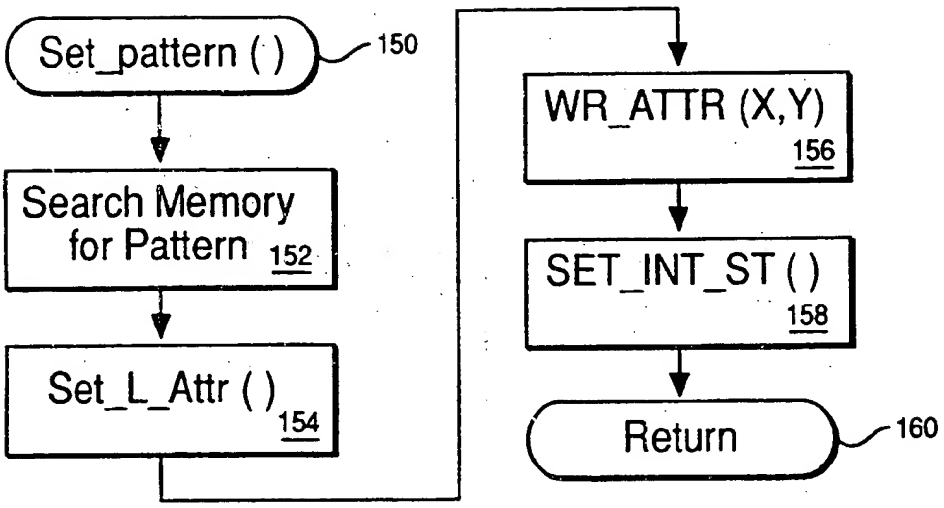

FIG. 7

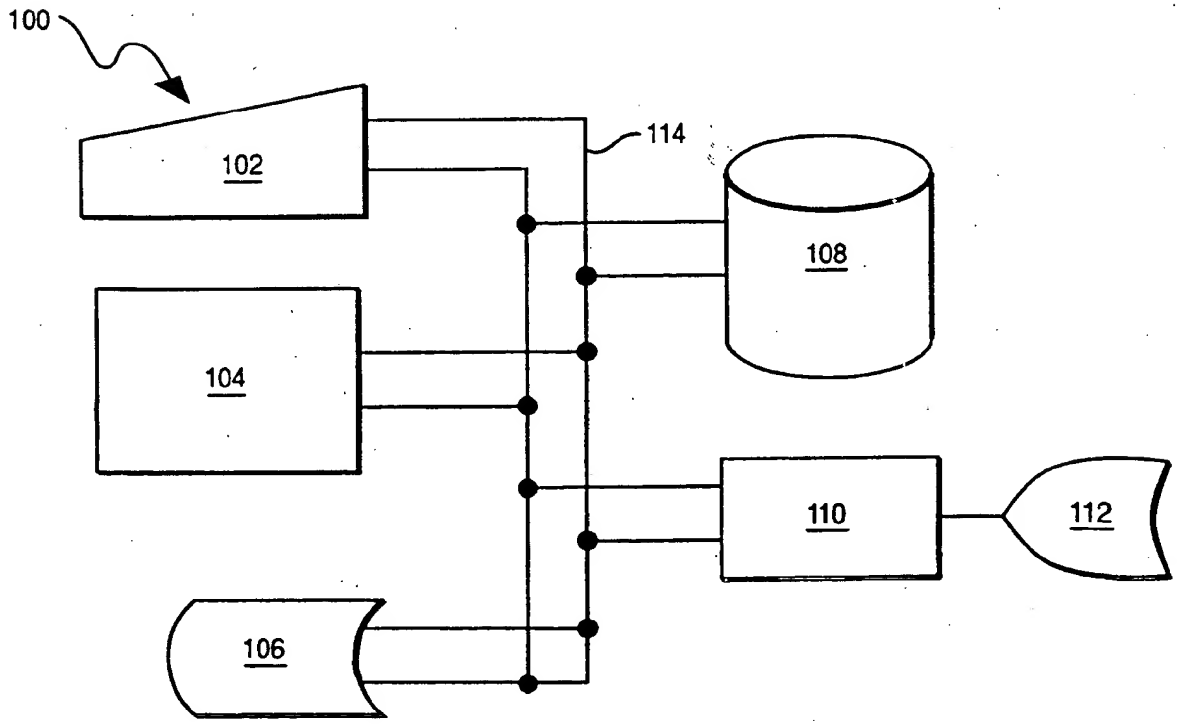

FIG. 9

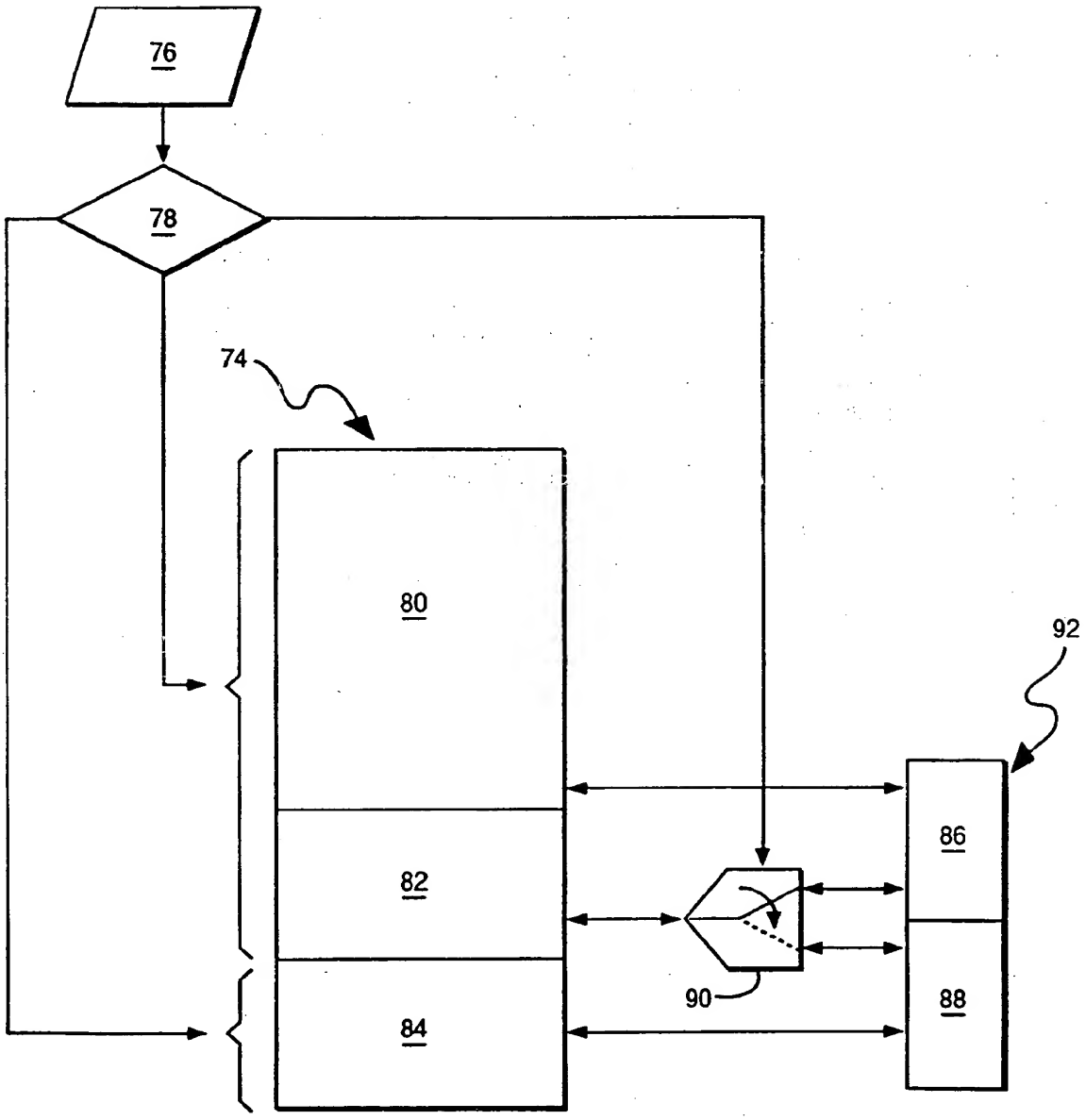

FIG. 8

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 92 30 6537

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
X A	US-A-4 875 186 (PRIME COMPUTER) * abstract; claims; figure 1 * * column 1, line 30 - line 40 *	1,2,5-8 9,10	G06F9/44
X	EP-A-0 169 565 (NEC) * abstract; figures * * page 4, line 1 - line 37 * * page 9, line 13 - page 12, line 3 *	1,2,5-8	
A	WO-A-8 401 635 (IBM) * page 3, line 21 - page 4, line 25 *	9,10	
A	J.L. BAER 'Computer Systems Architecture' 1980 , COMPUTER SCIENCE PRESS , ROCKVILLE, USA * page 355, line 15 - page 357, line 16 *	1-10	
			TECHNICAL FIELDS SEARCHED (Int. Cl.5) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	02 NOVEMBER 1992	GILL S.M.

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P0401)